AF/2192 [illegible]

DOCKET NO.: MSFT-0572 / 160077.01 **PATENT**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: **Scott Wiltamuth, Anders Hejlsberg, Peter F. Sollich, Bradley M. Abrams**	Confirmation No.: **5752**
Application No.: **09/900,111**	Group Art Unit: **2192**
Filing Date: **July 5, 2001**	Examiner: **Chameli Das**

For: **System and Methods for Providing Versioning of Software Components in a Computer Programming Language**

DATE OF DEPOSIT: August 17, 2005

I HEREBY CERTIFY THAT THIS PAPER IS BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE AS FIRST CLASS MAIL, POSTAGE PREPAID, ON THE DATE INDICATED ABOVE AND IS ADDRESSED TO THE COMMISSIONER FOR PATENTS, P.O. BOX 1450, ALEXANDRIA, VA 22313-1450.

[illegible]

TYPED NAME: Jeremiah J. Baunach
REGISTRATION NO.: 44,527

MS Appeal Brief - Patent
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF TRANSMITTAL
PURSUANT TO 37 CFR § 1.192

Transmitted herewith in triplicate is the APPEAL BRIEF in this application with respect to the Notice of Appeal received by The United States Patent and Trademark Office on **June 17, 2005**.

☐ Applicant(s) has previously claimed small entity status under 37 CFR § 1.27 .

☐ Applicant(s) by its/their undersigned attorney, claims small entity status under 37 CFR § 1.27 as:

- ☐ an Independent Inventor
- ☐ a Small Business Concern
- ☐ a Nonprofit Organization.

~~08/22/2005 GWORDOF1 00000025 233050 09900111~~

OIPE JC22 AUG 2 2 2005 PATENT & TRADEMARK OFFICE

DOCKET NO.: MSFT-0160 / 142385.01 **PATENT**

☐ Petition is hereby made under 37 CFR § 1.136(a) (fees: 37 CFR § 1.17(a)(1)-(4) to extend the time for response to the Office Action of to and through comprising an extension of the shortened statutory period of month(s).

	SMALL ENTITY		NOT SMALL ENTITY	
	RATE	FEE	RATE	FEE
☒ APPEAL BRIEF FEE	$250	$	$500	$500.00
☐ ONE MONTH EXTENSION OF TIME	$60	$	$120	$
☐ TWO MONTH EXTENSION OF TIME	$225	$	$450	$
☐ THREE MONTH EXTENSION OF TIME	$510	$	$1020	$
☐ FOUR MONTH EXTENSION OF TIME	$795	$	$1590	$
☐ FIVE MONTH EXTENSION OF TIME	$1080	$	$2160	$
☐ LESS ANY EXTENSION FEE ALREADY PAID	minus	($)	minus	($)
TOTAL FEE DUE	▨	$0	▨	$500.00

☒ The Commissioner is hereby requested to grant an extension of time for the appropriate length of time, should one be necessary, in connection with this filing or any future filing submitted to the U.S. Patent and Trademark Office in the above-identified application during the pendency of this application. The Commissioner is further authorized to charge any fees related to any such extension of time to Deposit Account 23-3050. This sheet is provided in duplicate.

☐ A check in the amount of **$ .00** is attached. Please charge any deficiency or credit any overpayment to Deposit Account No. 23-3050.

☒ Please charge Deposit Account No. 23-3050 in the amount of **$500.00**. This sheet is attached in duplicate.

☒ The Commissioner is hereby authorized to charge any deficiency or credit any overpayment of the fees associated with this communication to Deposit Account No. 23-3050.

Date: August 17, 2005

[illegible]
Jeremiah J. Baunach
Registration No. 44,527

Woodcock Washburn LLP
One Liberty Place - 46th Floor
Philadelphia PA 19103
Telephone: (215) 568-3100
Facsimile: (215) 568-3439

DOCKET NO.: MSFT-0572 / 160077.01 **PATENT**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:
Scott Wiltamuth, Anders Hejlsberg, Peter F. Sollich, Bradley M. Abrams

Confirmation No.: **5752**

Application No.: **09/900,111**

Group Art Unit: **2192**

Filing Date: **July 5, 2001**

Examiner: **Chameli Das**

For: **System and Methods for Providing Versioning of Software Components in a Computer Programming Language**

DATE OF DEPOSIT: August 17, 2005

I HEREBY CERTIFY THAT THIS PAPER IS BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE AS FIRST CLASS MAIL, POSTAGE PREPAID, ON THE DATE INDICATED ABOVE AND IS ADDRESSED TO THE COMMISSIONER FOR PATENTS, P.O. BOX 1450, ALEXANDRIA, VA 22313-1450.

[illegible]

TYPED NAME: Jeremiah J. Baunach
REGISTRATION NO.: 44,527

Mail Stop Appeal-Brief Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

APPELLANTS' BRIEF PURSUANT TO 37 C.F.R. § 1.192

This brief is being filed in support of Appellant's appeal from the rejections of claims 1-60 dated April 19, 2005. A Notice of Appeal was filed on June 17, 2005.

1.

REAL PARTY IN INTEREST

MICROSOFT CORPORATION is the real party in interest in the present application.

Originally, the inventors in the present application assigned their interests to MICROSOFT CORPORATION as recorded by the United States Patent and Trademark Office ("USPTO") on January 9, 2002 at Reel 012456, Frames 0033-0042.

2.

RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences

3.

STATUS OF CLAIMS

Claims 1-60 are pending in the present application. In summary of the outstanding Office Action, the specification is objected to for informalities. Claims 6, 28, 47 stand rejected under 35 USC § 112 as being allegedly indefinite. Claims 1-6, 12-13, 18-28, 34-35, 40-47, 53-54 and 59-60 stand rejected under 35 U.S.C. § 102(b) as allegedly anticipated by U.S. Patent No. 5,878,432 (Misheski). Claims 7-11, 14-17, 29-33, 36-39, 48-52 and 55-58 stand rejected under 35 U.S.C. § 103(a) as allegedly unpatentable over Misheski in view of one or more of the following: U.S. Patent No. 6,622,302 B1 (Delaney) and U.S. Patent No. 5,805,899 (Evans).

Claims 1-60 are reproduced in Appendix A, attached hereto, as they stand as of the date of this appeal.

4.

STATUS OF AMENDMENTS

No amendments to claims 1-60 have been filed subsequent to the final rejection.

5.

SUMMARY OF CLAIMED SUBJECT MATTER

The present invention relates generally to a system for versioning software components in connection with a computer programming language. In exemplary aspects, the versioning system makes use of intelligent defaults, provides a vehicle for unambiguous specification of developer intent and implements conflict resolution rules in connection with the versioning of software components. One way in which specification of developer intent is provided is by assigning a keyword to a software component. For example, virtual, new and override keywords enable clear specification of programmer intent, so that a programmer can apply preventative maintenance to the versioning problem in advance.

6.

ISSUES

A. Whether under 35 USC § 112, claims 6, 28, 47 are indefinite for improper use of trademarks and trade names in these claims.

B. Whether under 35 U.S.C. § 102(b), claims 1-6, 12-13, 18-28, 34-35, 40-47, 53-54 and 59-60 are unpatentable as being anticipated by Misheski.

C. Whether under 35 U.S.C. § 103(a), claims 7-11, 14-17, 29-33, 36-39, 48-52 and are unpatentable over Misheski in view of Delaney and/or Evans.

7. GROUPING OF CLAIMS

Claims 1-42 stand or fall together with respect to the rejections under 35 U.S.C. § 102 and 35 U.S.C. § 103. Each of these claims calls for the features (or related features rejected for the same reasons) of "specifying programmer intent with regard to versioning of said at least one software component by assigning at least one keyword to said at least one software component."

8. ARGUMENT

Regarding issue A above, Appellant respectfully traverses the Examiner's rejection of claims 6, 28, and 47 under 35 U.S.C. § 112 as allegedly indefinite for improper use of trademarks and trade names.

Rejections under 35 USC § 112

Claims 6, 28, 47 stand rejected under 35 USC § 112 as being allegedly indefinite for improper use of trademarks and trade names. According to the MPEP 2173.05(u) "The presence of a trademark or trade name in a claim is not, per se, improper under 35 U.S.C. 112, second paragraph, but the claim should be carefully analyzed to determine how the mark or name is used in the claim. It is important to recognize that a trademark or trade name is used to identify a source of goods, and not the goods themselves."

In view of the above, Claims 6, 28 and 47 have been amended to use the trademarks and alleged trade names in the proper manner by using them to identify the source of goods, and not the goods themselves. In response to the above, the final Office Action states in part "...the trade name is one of the element of the software component." First, claims 6, 28 and 47 do not speak of "elements" of software components. They state "...said at least one software component is at least one member of the object oriented programming language." Second, the trade names and/or trademarks are used to identify the source of the object oriented programming languages and not the languages themselves. Claims 6, 28 and 47 state "the object-oriented programming language is *from one of the sources of origin identified by* C#, Fortran, Pascal, Visual Basic, C, C++ and Java." Therefore, the use of trademark and/or trade names in claims 6, 28 and 47 is appropriate as identifying the source of goods, and not the goods themselves.

Therefore, reversal of the rejections under 35 U.S.C. § 112 for claims 6, 28 and 47 is earnestly solicited.

Regarding issue B above, Appellant respectfully traverses the Examiner's rejection of claims 1-6, 12-13, 18-28, 34-35, 40-47, 53-54 and 59-60 as allegedly unpatentable for being anticipated by Misheski.

Rejections under 35 USC § 102(b)

Claims 1-6, 12-13, 18-28, 34-35, 40-47, 53-54 and 59-60 stand rejected under 35 U.S.C. § 102(b) as allegedly anticipated by Misheski.

The Law of Patent Claim Construction

In order to determine whether a claim is anticipated under 35 U.S.C. § 102, the limitations of the claim must be construed so that they can be compared with the prior art. Thus, an inquiry as to whether the prior art anticipates a claim requires a construction of the claim in question *Trintec Indus. v. Top-U.S.A. Corp.*, 295 F.3d 1292, 1295, 63 U.S.P.Q.2d 1597 (Fed. Cir. 2002). We set forth below the law relating to the construction of claims.

The words used in the claims define the scope of a patented invention. *Phillips v. AWH Corp.*, 2005 U.S.App.LEXIS 13954 at *20 (Fed. Cir. Jul. 12, 1995) (en banc). Thus, a court should look first to the language of the claims. *K2-Corp. v. Solomon S.A.*, 191 F.3d 1356, 1362 (Fed. Cir. 1999) ("We begin, of course, with the language of the claims."); *Digital Biometrics, Inc. v. Identix, Inc.*, 149 F.3d 1335, 1344 (Fed. Cir. 1998) ("To determine the proper meaning of claims we first consider the so-called intrinsic evidence, i.e., the claims, the written description, and, if in evidence, the prosecution history."). The claim terms themselves "provide substantial guidance as to the meaning of particular claim terms," and "the context in which a term is used in the asserted claim can be highly instructive." *Phillips*, 2005 U.S.App.LEXIS 13954 at *27. Moreover, because "claim terms are normally used consistently throughout the patent, the usage of a term in one claim can often illuminate the meaning of the same term in other claims." *Id.* at *28.

The words used in a claim "are generally given their ordinary and customary meaning." *Id.* at *22 (quoting *Vitronics Corp. v. Conceptronic, Inc.*, 90 F.3d 1576, 1582 (Fed. Cir. 1996)). The "ordinary and customary meaning" of a claim term is "the meaning that the term would have to a person of ordinary skill in the art in question at the time of the invention." *Phillips*, 2005 U.S.App.LEXIS 13954 at *22. When determining what meaning a person of ordinary skill in the relevant art would attach to a particular claim term, that person "is deemed to read the claim term not only in the context of the particular claim in which the disputed term appears, but in the context of the entire patent, including the specification." *Id.* at *24; *see also Multiform Desiccants, Inc. v. Medzam Ltd.*, 133 F.3d 1473, 1478 (Fed. Cir. 1998). A court, therefore, should "rely heavily on the written description for guidance as to the meaning of the claims." *Phillips*, 2005 U.S.App.LEXIS 13954 at *35. It has thus been said that "the 'ordinary meaning' of a claim term is the meaning to the ordinary artisan after reading the entire patent." *Id.* at *48-*49.

In determining the ordinary meaning of a claim term, a dictionary may be used. *Id.* at *53 ("Dictionaries or comparable sources are often useful to assist in understanding the commonly understood meaning of words"). However, the dictionary should not be relied on to attribute an "abstract" meaning to a term without regard to that term's meaning in the context of the specification or other intrinsic evidence. *Id.* at *49.

While it is often said that one cannot read a limitation into a claim from the specification, it is entirely proper to define terms already in the claims by their usage in the specification. In particular, the inventor may act as his own "lexicographer," and, if the specification defines a term in a manner "that differs from the meaning [the term] would otherwise possess ... the inventor's lexicography governs." *Id.* at *33-*34.

The prosecution history (sometimes called the file wrapper) of a patent can also be important in determining the meaning of a disputed claim term. *See id.* at *35-*37; *Markman*, 52 F.3d at 980; *Southwall Tech., Inc. v. Cardinal IG Co.*, 54 F.3d 1570, 1576 (Fed. Cir.), *cert. denied*, 516 U.S. 987 (1995) ("[T]he prosecution history, as well as the specification and other claims, must be examined to determine the meaning of terms in the claims.") Indeed, the prosecution history "is often of critical significance in determining the meaning of the claims" and arguments made by the applicant and his attorney to distinguish prior art to obtain the patent are critical to a proper interpretation of the patent claims. *Vitrionics Corp. v. Conceptronic, Inc.*, 90 F.3d 1576, 1582 (Fed. Cir. 1996). Further, unless the prosecution history reveals another purpose, there is a rebuttable presumption that a claim amendment that narrows the scope of a claim was made with the intent to surrender all subject matter between the broader and narrower versions of the claim. *Festo Corp. v. Shoketsu Kinzoku Kogyo Kabushiki Co.*, 122 S. Ct. 1831, 1841-42, 152 L. Ed. 2d 944, 62 U.S.P.Q.2d 1705 (2002) (citing *Warner-Jenkinson Company, Inc. v. Hilton Davis Chemical Co.*, 117 S.Ct. 1040, 41 U.S.P.Q.2d 1873 (1997)).

Finally, a court may also refer to "extrinsic" evidence where the court deems it useful in construing undefined or ambiguous terms in the patent claims. *Digital Biometrics*, 149 F.3d at 1344 ("[C]onsideration of extrinsic evidence may be necessary to determine the proper construction."). Extrinsic evidence in general, and expert testimony in particular, may be used only to help the court come to the proper understanding of the claims; it may not be used to vary or contradict the claim language. *Vitrionics Corp. v. Conceptronic, Inc.*, 90 F.3d 1576, 1584 (Fed. Cir. 1996); *Phillips*, 2005 U.S.App.LEXIS 13954 at *41-*42 ("extrinsic evidence may be useful to the court, but it is unlikely to result in a reliable interpretation of patent claim scope unless considered in the context of the intrinsic evidence").

Claim 1

With respect to claim 1, the previous 12/09/2004 Office Action contends that Misheski teaches "specifying programmer intent with regard to versioning of said at least one software component by assigning at least one keyword to said at least one software component." The Office Action is supporting this contention by stating Misheski uses extensible classes which can be modified or customized by the programmer to implement a source code repository environment. However, no mention is made of assigning a keyword. Appellants respectfully submit that using extensible classes is not the same as assigning a keyword to a software component. Thus, for at least the reasons above, Appellants submit that all the limitations of claim 1 are not taught or suggested by Misheski.

In response to the above, the final Office Action dated 4/19/2005 states "The extensible classes are extended by a programmer to provide this versioning support (Abstract, lines 10-14). This extension or customization are performed by 'overriding' the default operation (col 9, lines 1-5), where 'overriding' is considered as a keyword." However, Misheski does not in fact consider "overriding" a keyword, nor does it teach assigning keywords for overriding or other operations to specify programmer intent. The way Misheski describes performing overriding is by extending or customizing the default operations by "redefinition," within the subclasses as described in the excerpt below from Misheski:

> "Deafault operations are used to provide generic function to subclasses. The subclasses can simply use the default operations or they can customize or extend the default operations by redefinition. Redefinition of a default operation is called overriding the default operation." Col. 9, lines 1-5.

Nowhere does Misheski teach using a keyword to indicate that certain operations or methods are to be overridden. According to *Phillips*, 2005 U.S.App.LEXIS 13954 at *35, a court should "rely heavily on the written description for guidance as to the meaning of the claims." Thus, for the meaning of "assigning a keyword," looking to Appellants' specification, it is described as a specific word used in the declaration of the method. For example, page 19, lines 13-15 provide:

> "Still further, SecondParty may decide for any reason that B 412's G 414a should override A 402b's G 414b. This intent can be specified by using the override keyword 442 in the declaration of G 414b, as shown in Fig. 4E."

Fig. 4E of Appellants' specification also shows using the keyword "override" in the declaration of the method G() 414b. Although Misheski describes subclasses that override default operations, Misheski does not teach using a specific word in the declaration of the method or operation to indicate that it should be overridden. Therefore, Appellants respectfully submit that all the limitations of claim 1 are not taught or suggested by Misheski for at least the reasons presented above.

Regarding claims 2-6, 12-13, 18-28, 34-35, 40-47, 53-54 and 59-60, they either depend directly or indirectly from claim 1, and thus include all of the limitations of claim 1, or were rejected for the same reasons as claim 1. Therefore, Appellants respectfully submit that all the limitations of claims 2-6, 12-13, 18-28, 34-35, 40-47, 53-54 and 59-60 are not taught or suggested by Misheski for the same reasons presented above.

Therefore, reversal of the rejections under 35 U.S.C. § 102 for claims 2-6, 12-13, 18-28, 34-35, 40-47, 53-54 and 59-60 is earnestly solicited.

Regarding issue C above, Appellant respectfully traverses the Examiner's rejection of claims 7-11, 14-17, 29-33, 36-39, 48-52 as allegedly unpatentable over Misheski in view of Delaney and/or Evans.

Rejections under 35 USC § 103(a)

Claims 7-11, 14-17, 29-33, 36-39, 48-52 and 55-58 stand rejected under 35 U.S.C. § 103(a) as allegedly unpatentable over Misheski in view of Delaney and/or Evans.

Claims 7-11, 14-17, 29-33, 36-39, 48-52 and 55-58, either depend directly or indirectly from claim 1, and thus include all of the limitations of claim 1, or were rejected for the same reasons as claim 1. Therefore, Appellants respectfully submit that all the limitations of claims 7-11, 14-17, 29-33, 36-39, 48-52 and 55-58 are not taught or suggested by Misheski, Delaney, Evans or any combination thereof, for at least the reasons presented above for claim 1.

"To establish prima facie obviousness of a claimed invention, all the claim limitations must be taught or suggested by the prior art." MPEP § 2142. Since all the limitations of claims 7-11, 14-17, 29-33, 36-39, 48-52 and 55-58 are not taught or suggested by Misheski, Delaney, Evans or any combination thereof, for at least the reasons presented above, reversal of the rejections under 35 U.S.C. § 103(a) for claims 17-11, 14-17, 29-33, 36-39, 48-52 and 55-58 is earnestly solicited.

Conclusion

Appellants thus submit that claims 1-60 patentably define over Misheski in view of Delaney and Evans, or any other reference of record, taken alone or in combination. Appellants also submit that claims 6, 28 and 47 comply with 35 U.S.C. § 112. For all the foregoing reasons, Appellants respectfully request that the Board reverse the rejections of claims 1-60.

Respectfully submitted,

Date: August 17, 2005

[illegible]

Jeremiah J. Baunach
Registration No. 44,527

Woodcock Washburn LLP
One Liberty Place - 46th Floor
Philadelphia PA 19103
Telephone: (215) 568-3100
Facsimile: (215) 568-3439

APPENDIX A

Claims on Appeal

1. (original) A method for providing versioning support for at least one software component of an object-oriented programming language, the method comprising:

specifying programmer intent with regard to versioning of said at least one software component by assigning at least one keyword to said at least one software component.

2. (original) A method according to claim 1, wherein said assigning said at least one keyword includes assigning at least one of virtual, new and override keywords.

3. (original) A method according to claim 1, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component overrides another software component.

4. (original) A method according to claim 1, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component is capable of being overridden by another software component.

5. (original) A method according to claim 1, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component hides another software component.

6. (previously presented) A method according to claim 1, wherein said at least one software component is at least one member of the object-oriented programming language and the object-oriented programming language is from one of the sources of origin identified by C#, Fortran, Pascal, Visual Basic, C, C++ and Java.

7. (original) A method according to claim 1, wherein said specifying of programmer intent includes assigning intelligent defaults to said at least one software component in the absence of assigning said at least one keyword to said at least one software component.

8. (original) A method according to claim 7, wherein when programmer intent is not fully specified, the compiler of the programming language produces a warning before assigning said intelligent defaults.

9. (original) A method according to claim 7, wherein said assigning of intelligent defaults includes assigning to said at least one software component the most limited form of accessibility, based upon the type of said at least one software component.

10. (original) A method according to claim 7, wherein by default, when said at least one software component is at least one method declaration with no accessibility modifiers appearing in the corresponding class, the at least one method declaration is defaulted to be private to that class.

11. (original) A method according to claim 7, wherein by default, said at least one software component is non-virtual, rather than virtual.

12. (original) A method according to claim 1, wherein said specifying of programmer intent includes providing a versioning-aware overload resolution method to locate a second method invoked by a first method invocation.

13. (original) A method according to claim 12, wherein said versioning-aware overload resolution method includes:

determining the type indicated by the first method invocation,

checking up the inheritance chain until at least one applicable, accessible, non-override method declaration is found;

performing overload resolution on the set of applicable, accessible, non-override methods declared for the type; and

selecting the second method based on the performance of said overload resolution.

14. (original) A method according to claim 12, wherein for a virtual method invocation, said versioning-aware overload resolution method includes determining the second method based on the run-time type of the instance of the first method invocation.

15. (original) A method according to claim 12, wherein for a non-virtual method invocation, said versioning-aware overload resolution method includes determining the second method based on the compile-time type of the instance of the first method invocation.

16. (original) A method according to claim 12, wherein said versioning-aware overload resolution method includes bounding names at run-time, and not bounding offsets at compile-time.

17. (original) A method according to claim 12, wherein the overload resolution method prevents a base software component from breaking the functionality of a derived software component when versioning the base software component and such breaking is not intended by the programmer.

18. (original) A method according to claim 1, wherein said at least one software component is binary compatible with code utilizing other versions of said at least one software component.

19. (original) A method according to claim 1, wherein said at least one software component is source compatible with code utilizing other versions of said at least one software component.

20. (original) A computer readable medium bearing computer executable instructions for carrying out the method of claim 1.

21. (original) A modulated data signal carrying computer executable instructions for performing the method of claim 1.

22. (original) A computing device comprising means for performing the method of claim 1.

23. (original) A computer readable medium having stored thereon a plurality of computer-executable modules written in an object-oriented programming language, the computer executable modules comprising:

a versioning mechanism enabling a programmer to specify intent with regard to versioning of at least one software component by assigning at least one keyword to said at least one software component.

24. (original) A computer readable medium according to claim 23, wherein said assigning said at least one keyword includes assigning at least one of virtual, new and override keywords.

25. (original) A computer readable medium according to claim 23, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component overrides another software component.

26. (original) A computer readable medium according to claim 23, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component is capable of being overridden by another software component.

27. (original) A computer readable medium according to claim 23, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component hides another software component.

28. (previously presented) A computer readable medium according to claim 23, wherein said at least one software component is at least one member of the object-oriented programming language and the object-oriented programming language is from one of the sources of origin identified by C#, Fortran, Pascal, Visual Basic, C, C++ and Java.

29. (original) A computer readable medium according to claim 23, wherein said specifying of programmer intent includes assigning intelligent defaults to said at least one

software component in the absence of assigning said at least one keyword to said at least one software component.

30. (original) A computer readable medium according to claim 29, wherein when programmer intent is not fully specified, the compiler of the programming language produces a warning before assigning said intelligent defaults.

31. (original) A computer readable medium according to claim 29, wherein said assigning of intelligent defaults includes assigning to said at least one software component the most limited form of accessibility, based upon the type of said at least one software component.

32. (original) A computer readable medium according to claim 29, wherein by default, when said at least one software component is at least one method declaration with no accessibility modifiers appearing in the corresponding class, the at least one method declaration is defaulted to be private to that class.

33. (original) A computer readable medium according to claim 29, wherein by default, said at least one software component is non-virtual, rather than virtual.

34. (original) A computer readable medium according to claim 23, wherein said specifying of programmer intent includes providing a versioning-aware overload resolution method to locate a second method invoked by a first method invocation.

35. (original) A computer readable medium according to claim 34, wherein said versioning-aware overload resolution method includes:

determining the type indicated by the first method invocation,

checking up the inheritance chain until at least one applicable, accessible, non-override method declaration is found;

performing overload resolution on the set of applicable, accessible, non-override methods declared for the type; and

selecting the second method based on the performance of said overload resolution.

36. (original) A computer readable medium according to claim 34, wherein for a virtual method invocation, said versioning-aware overload resolution method includes determining the second method based on the run-time type of the instance of the first method invocation.

37. (original) A computer readable medium according to claim 34, wherein for a non-virtual method invocation, said versioning-aware overload resolution method includes determining the second method based on the compile-time type of the instance of the first method invocation.

38. (original) A computer readable medium according to claim 34, wherein said versioning-aware overload resolution method includes bounding names at run-time, and not bounding offsets at compile-time.

39. (original) A computer readable medium according to claim 34, wherein the overload resolution method prevents a base software component from breaking the functionality of a derived software component when versioning the base software component and such breaking is not intended by the programmer.

40. (original) A computer readable medium according to claim 23, wherein said at least one software component is binary compatible with code utilizing other versions of said at least one software component.

41. (original) A computer readable medium according to claim 23, wherein said at least one software component is source compatible with code utilizing other versions of said at least one software component.

42. (original) An object-oriented programming language for producing computer executable modules, comprising:

a versioning mechanism enabling a programmer to specify intent with regard to versioning of at least one software component by assigning at least one keyword to said at least one software component.

43. (original) An object-oriented programming language according to claim 42, wherein said assigning said at least one keyword includes assigning at least one of virtual, new and override keywords.

44. (original) An object-oriented programming language according to claim 42, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component overrides another software component.

45. (original) An object-oriented programming language according to claim 42, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component is capable of being overridden by another software component.

46. (original) An object-oriented programming language according to claim 42, wherein said assigning said at least one keyword to said at least one software component specifies programmer intent with regard to whether said at least one software component hides another software component.

47. (previously presented) An object-oriented programming language according to claim 42, wherein said at least one software component is at least one member of the object-oriented programming language and the object-oriented programming language is from one of the sources of origin identified by C#, Fortran, Pascal, Visual Basic, C, C++ and Java.

48. (original) An object-oriented programming language according to claim 42, wherein said specifying of programmer intent includes assigning intelligent defaults to said at least one software component in the absence of assigning said at least one keyword to said at least one software component.

49. (original) An object-oriented programming language according to claim 48, wherein when programmer intent is not fully specified, the compiler of the programming language produces a warning before assigning said intelligent defaults.

50. (original) An object-oriented programming language according to claim 48, wherein said assigning of intelligent defaults includes assigning to said at least one software component the most limited form of accessibility, based upon the type of said at least one software component.

51. (original) An object-oriented programming language according to claim 48, wherein by default, when said at least one software component is at least one method declaration with no accessibility modifiers appearing in the corresponding class, the at least one method declaration is defaulted to be private to that class.

52. (original) An object-oriented programming language according to claim 48, wherein by default, said at least one software component is non-virtual, rather than virtual.

53. (original) An object-oriented programming language according to claim 42, wherein said specifying of programmer intent includes providing a versioning-aware overload resolution method to locate a second method invoked by a first method invocation.

54. (original) An object-oriented programming language according to claim 53, wherein said versioning-aware overload resolution method includes:

determining the type indicated by the first method invocation,

checking up the inheritance chain until at least one applicable, accessible, non-override method declaration is found;

performing overload resolution on the set of applicable, accessible, non-override methods declared for the type; and

selecting the second method based on the performance of said overload resolution.

55. (original) An object-oriented programming language according to claim 53, wherein for a virtual method invocation, said versioning-aware overload resolution method includes determining the second method based on the run-time type of the instance of the first method invocation.

56. (original) An object-oriented programming language according to claim 53, wherein for a non-virtual method invocation, said versioning-aware overload resolution

method includes determining the second method based on the compile-time type of the instance of the first method invocation.

57. (original) An object-oriented programming language according to claim 53, wherein said versioning-aware overload resolution method includes bounding names at run-time, and not bounding offsets at compile-time.

58. (original) An object-oriented programming language according to claim 53, wherein the overload resolution method prevents a base software component from breaking the functionality of a derived software component when versioning the base software component and such breaking is not intended by the programmer.

59. (original) An object-oriented programming language according to claim 42, wherein said at least one software component is binary compatible with code utilizing other versions of said at least one software component.

60. (original) An object-oriented programming language according to claim 42, wherein said at least one software component is source compatible with code utilizing other versions of said at least one software component.